Predicting English Premier League Games from Match Statistics

Machine Learning Final Project Part 2

Bhavik Naik
Faculty of Business and I.T
Ontario Tech University
Oshawa Ontario Canada
bhavik.naik@ontariotechu.net

1 Frame the Problem

Sports is one of the most popular hobbies in the world. Athletes train daily to stay ahead of their competition while fans enjoy watching the action unfold on live TV. The Premier League in England is the most watched football league in the world [SportMob, 2021]. Players from around the world compete against 20 teams to be claimed the best English football side in the country. With 925.3 million live streaming minutes and an average of 462,000 viewers per match in the 2019/20 season at NBC Sports Network, the Premier League's fan base extends further than England itself [SportsPro, 2020]. Super computers during the beginning and middle of the season always makes headlines as these models predict the final league standings based on the information that has already been collected. Nowadays, there is a lot of data collected per game. From the basic numbers such as shots on goal and kilometers run, to more advanced numbers such as the expected goal (XG) which takes in player positioning and other data to calculate the chance of goal that the team has. Teams and managers have altered their playing styles to improve this number so that the possibility of a goal scoring chance is higher. By recording all these data points, we can utilize regression algorithms to calculate the results based on the key factors that influence the game. If we are successfully able to find a model that can take in data points and accurately calculate the match result, we can theoretically predict what could happen in future games. The dataset that was chosen for this paper is the "English Premier League stats 2019-2020" which contain 45 columns of data corresponding to each game, respective to a single team. The XG, scored, conceded, pressing plays, number of shots, betting odds, and more are contained in this dataset. In this paper, we will train three machine learning models to take in this data and predict the correct results. We will be using a deep neural network, linear regression, and SGD regressor as our three machine learning models. We will explore the data first, clean up the data, and finally, showcase the results from all three. The best performing algorithm will get graphs depicting the results.

2 Gain Insights

In this section, we will provide four graphs that try to gain insights into the data.

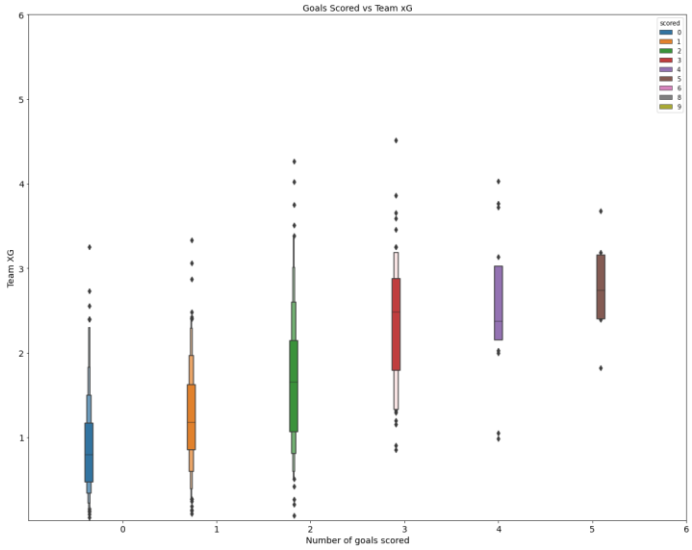

Figure 1 The correlation between goals scored (x-axis) and the team xG. We can see that the higher the xG, the more goals are being scored by the team.

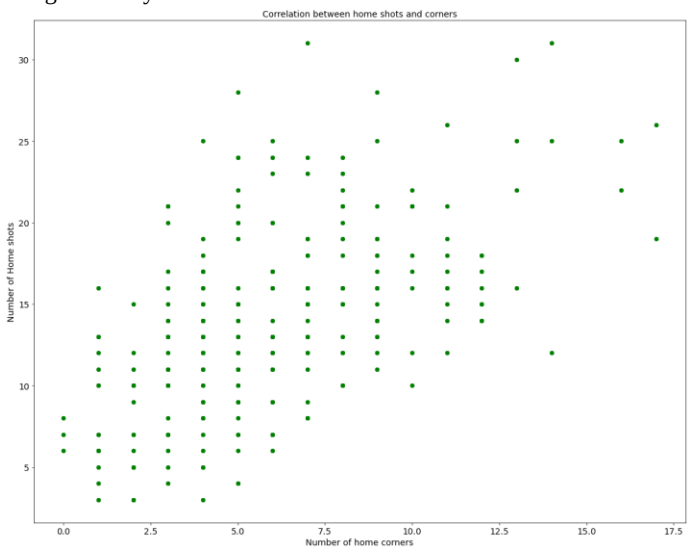

Figure 2 The number of home corners (x-axis) vs the number of shots. As we can see, the more corners a team has, the more shots they have on average. It is quite interesting to see correlations in areas that you would not expect.

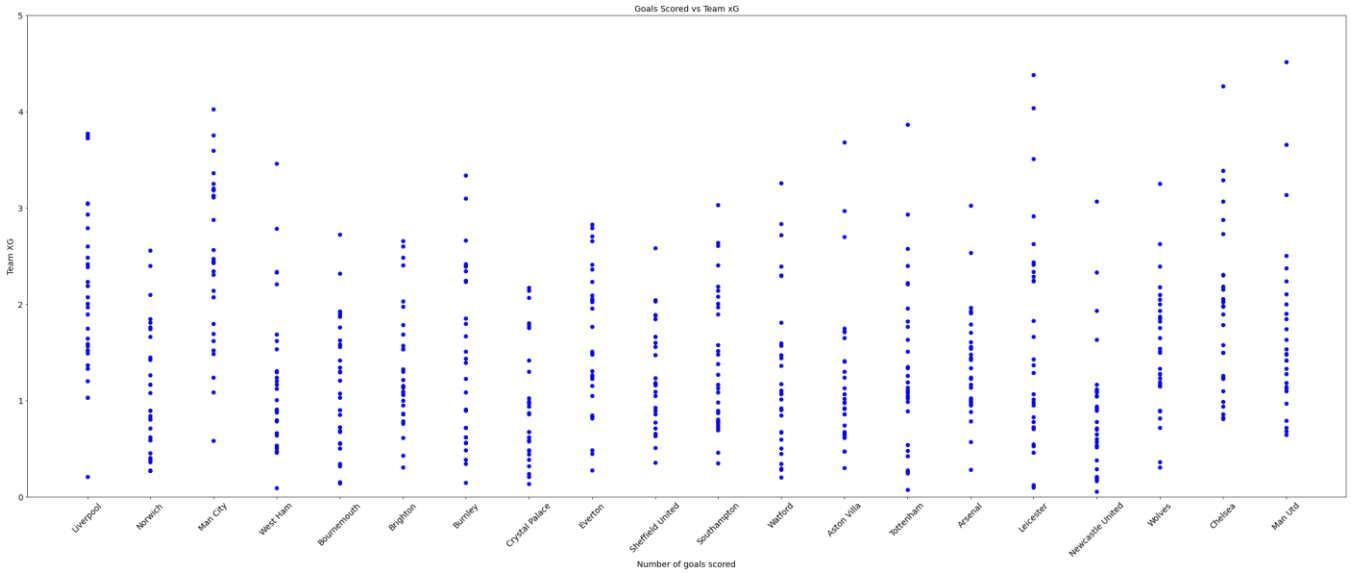

Figure 3 A scatter plot showing the team's xG scores across their matches. We can see that higher ranked teams like Liverpool, Chelsea, and Manchester City have higher xG scores because they attack more than lower ranked teams like Newcastle United.

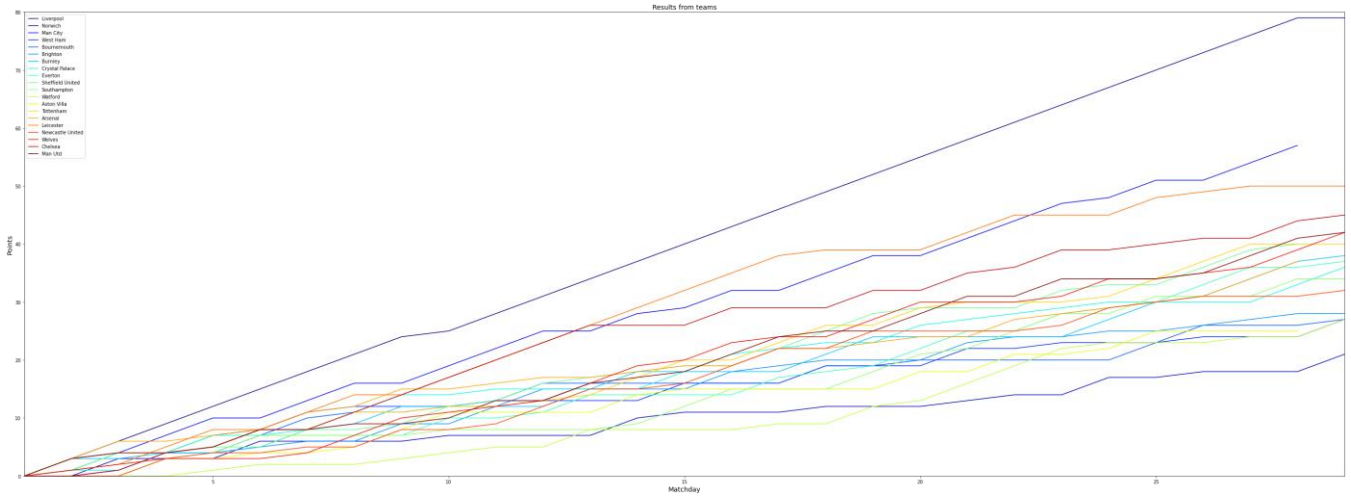

Figure 4 The results from each game, mapped over the entire season. We can see every team and how they were able to accumulate points throughout the season.

3 Prepare the Data

The information that was present in the dataset did not include any null values which prevented us to remove lines that contained missing data. There were numerous columns that we had to drop which did not make sense in calculating the result, such as the referee name, matchday (the day that the match was on), date, match time, round (the number of played matches by the team in the season), points (the number of points that team gained after the match), total points (the number of points that team has coming into the match). All the bet odds were also dropped which we felt did not contribute to the result of the game. There were some columns that were dropped that included numbers which could help the algorithm, but we felt that these numbers did not further enhance the result of the algorithm. Those include, npxG and npxA (the expected goal value without penalties, where each penalty adds 0.76 to the XG), and npxGD (the difference XG between the teams). In terms of the data, we had to convert some of the columns to integers, such as the home and away column, and the result column. The values were replaced from *h (home)* and *a (away)* to *0* and *1*. While in the results column, *w (win)* was represented as 2, *d (draw)* was represented as *1*, and *l (lost)* was represented as *0*. We also converted the 'teamId' field which contains the team's name to integers. Each team got a number from 0 to 19 and this was inserted into the data frame in expense of the name of the team. We decided to keep in team name because higher ranked teams perform better than their counterparts. We then used the SKLearn *train_test_split* function to split the data into training and testing tests using a 60-40% split. We set the test column to the results column as this is the column that we are trying to predict.

4 Machine Learning Algorithms

In this paper, we used three machine learning algorithms to try to predict the results based on the data in the column. We will be looking at a deep neural network (DNN) that is based of Keras, a linear regression algorithm, and a Stochastic gradient descent (SGD) regressor to evaluate the data and predict results. We will

be comparing the three algorithms using mean squared error and mean absolute error. Table 1 will showcase the comparative performance table which will contain the results from each algorithm.

4.1 Deep Neural Network

Starting off with the deep learning algorithm, we created a Keras sequential model that uses seven layers. It starts off with a flatten layer that takes in the 29 columns, then a 128-node connected layer with activation set to "relu" and kernel initializer set to normal. The third layer is a dropout layer with 30% of nodes dropping out. Then came another 128-node connected layer with the kernel initializer set to normal and activation set to "relu". A drop out layer with a 50% chance came after and the sixth layer had a 64-node connected dense layer with kernel initializer set to normal and activation set to "relu". Finally, the last layer was a 3-node dense with the activation set to linear. We used mean squared error as the loss and the metrics we added include the mean squared error and the mean absolute error. The optimizer we used was "adam". When training it, we set epochs to 100 and fit it with the training set while the validation set was set to the testing set.

4.2 Linear Regression

The second algorithm that we used to predict the results was a linear regression algorithm. A linear regression algorithm is one the most popular machine learning models to use when dealing with regression problems. We fit the testing data on it and left the parameters as the default depending on what is given by Scikit-learn.

4.3 SGD Regressor

The final machine learning model that we used is a Stochastic gradient descent or SGD regressor. The SGD regressor is also one of the popular regression algorithms and it is tailored to a small amount of data. We used the SkLearn StandardScaler function to scale up the testing data, and then, we called the SGDRegressor with all its default values. After training it on the training dataset, we predicted the results of the testing dataset.

4.4 Results

We will now discuss the results from the three algorithms.

4.4.1 Deep Neural Network. For the results, the DNN gave us a score of 0.0150 for the mean squared error while the mean absolute error was 0.0852. The lower the better when it comes to these two metrics, and we can see that the deep neural network performed fairly well as it was able to keep both metrics under 1.

4.4.2 Linear Regression. The linear regression performed much better than the deep learning model. The mean squared error was 1.09e-29 and the mean absolute error was 2.58e-15. This shows that the algorithm got close to 0, which means that it was able to predict the result nearly precisely from the test data that it received.

4.4.3 SGD Regressor. The SGD regressor gave us a mean squared error of 0.0024037 and a mean absolute error of 0.0340784. It also performed well as both values are significantly below 1.

Algorithm	***Mean Squared Error (MSE)***	***Mean Absolute Error (MAE)***
Deep Neural Network	0.0150	0.0852
Linear Regression	1.09 e-29	2.58 e-15
SDG Regressor	0.0024037	0.0340784

Table 1: The comparative performance table

As we can see from table 1, the linear regression algorithm performed the best and got the closest to predicting the testing data. The SDG regressor was second and then the deep neural network performed the worst.

5 Best Performing Algorithm

In this section, we will show four graphs that showcase the results of the linear regression model as this is the best performing algorithm.

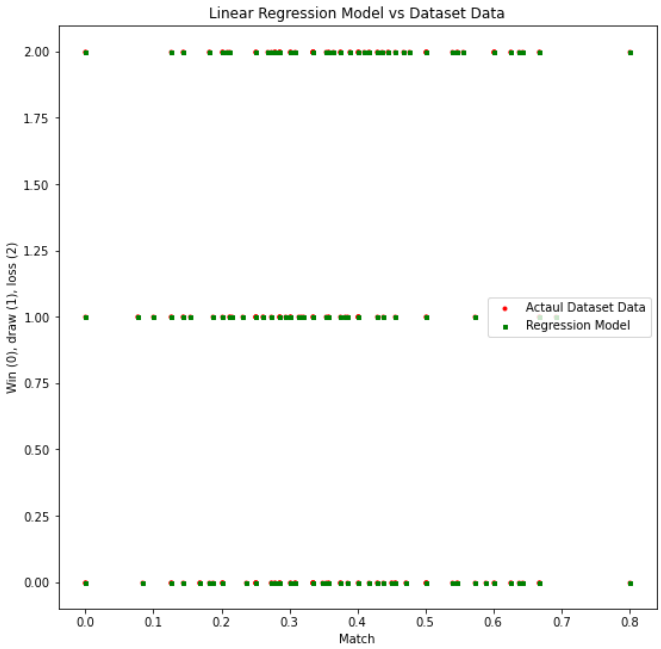

Figure 5 *The first graph shows a scatter plot of the testing data (red) vs the regression model (green). As we can see, the points are similar, almost overlapping. It is difficult to see the red dots as the green ones are directly on top of them. On the y axis, 0 represents a loss , 1 represents a draw and 2 represents a win*

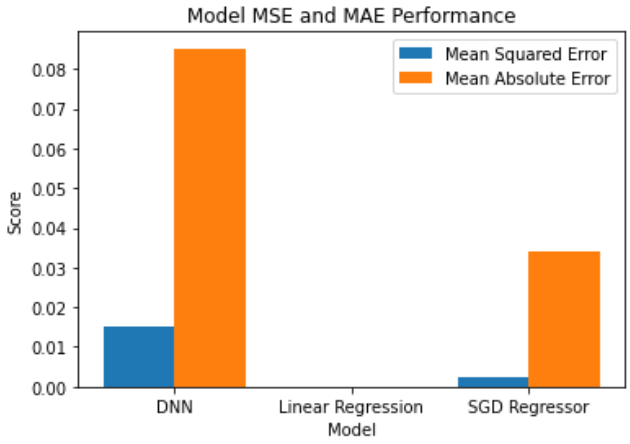

Figure 6 The MSE and MAE performance across the models. As we can see, the linear regression model did so well that it does not fit in the scale when comparing the other 2 models

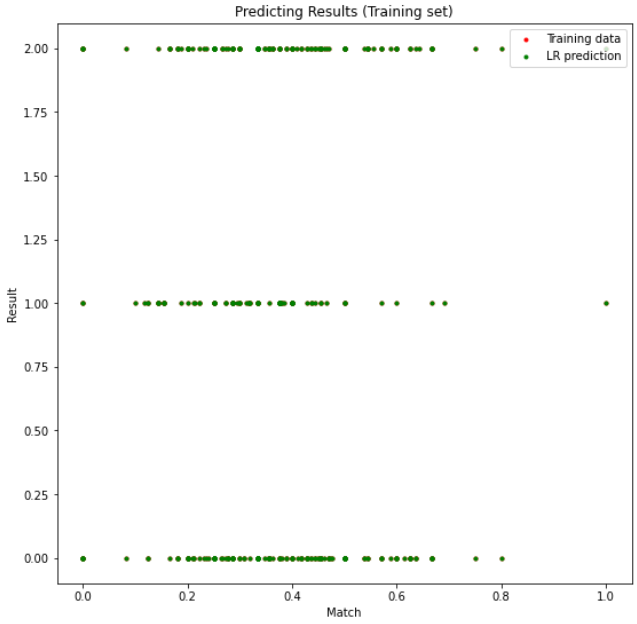

Figure 7 This figure is similar to figure 5 but instead of showing the performance of the linear regression model in terms of the testing data, we look at how well it can predict the training data. Based on this graph, we can see that it easily is able to predict the training data too.

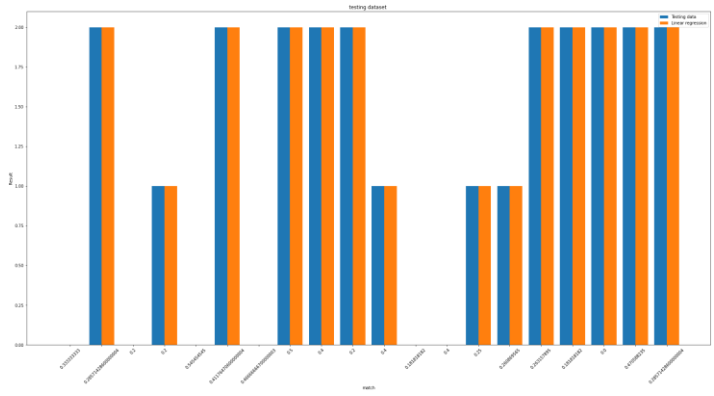

Figure 8 This bar graph shows how similar the linear regression model is to the testing data. We can see that the orange bar (linear regression) is exactly the same as thes testing data. This is just a slice of the data as graphing all of the data will not fit.

APPENDIX

In the appendix, we will show all of the Python code that we used to achieve the results of the report.

```
# Bhavik Naik
# Machine learning final project

# Import the various libaries
import pandas as pd
import os
import math
import numpy as np
import tensorflow as tf
from tensorflow import keras
import keras.optimizers
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import plotly.graph_objects as go
import plotly.express as px
from google.colab import files
import io
uploaded = files.upload()
# link to the kaggle
#https://www.kaggle.com/idoyo92/epl-stats-20192020
# read in the csv
df = pd.read_csv(io.BytesIO(uploaded['epl2020.csv']),  index_col=0)

# replace the home and away to numbers, and win, lost and draw
df = df.replace({'h': 0, 'a': 1})
df = df.replace({'w': 2, 'd': 1, 'l': 0})
df.head()
# create a dictionay with the team results over the reason to graph it (figure 4)
teamresults = {}
teamresults2 = {}
for i in df['teamId']:
  teamresults[i] = list((df.loc[df['teamId'] == i, 'pts']))
for i in teamresults:
  teamresults2[i] = []
for i in teamresults:
  for j in range(len(teamresults[i])):
    teamresults2[i].append(sum(teamresults[i][:j]))
# plot the lines on the graph, with the results
f1 = plt.figure(figsize=(50, 18))
# set the colour
plt.gca().set_prop_cycle(plt.cycler('color', plt.cm.jet(np.linspace(0, 1, 20))))
labels = []
for i in teamresults2:
  plt.plot(range(1, len(teamresults2[i]  )+ 1), teamresults2[i])
  labels.append(i)
plt.legend(labels, loc="upper left")
plt.xlim([1, 29])
plt.ylim([0, 80])
plt.xlabel("Matchday", fontsize="14")
plt.ylabel("Points", fontsize="14")
plt.title("Results from teams", fontsize="14")

# create the graph for the teamid and the xg for the team (figure 3)
f3 = plt.figure(figsize=(40, 15))
```

```
plt.scatter(x = df['teamId'] , y = df['xG'], color="blue")
plt.xticks(fontsize=14, rotation=45)
plt.yticks(fontsize=14)
plt.xlabel(xlabel="Number of goals scored", fontsize=14)
plt.ylabel(ylabel="Team XG", fontsize=14)
plt.title("Goals Scored vs Team xG", fontsize=14)
plt.ylim([0, 5])

# replace the teamid names to integers. use a dictionary to do this
a = 0
values = {}
for i in df['teamId']:
  if i not in values:
    values[i] = a
    a += 1
# replace the teamid with the integers
df = df.replace(values)

# drop all of the axis we dont need
df = df.drop(["Referee.x", "matchDay", "date", "matchtime", "pts", "tot_points", "round",
              "matchtime", "npxGD", "xpts", "npxG", "npxGA",
              "B365H.x", "B365D.x", "B365A.x"], axis=1)

import seaborn as sns

# create the correlation graph between home corners and home shots (figure 2)
f2 = plt.figure(figsize=(19, 15))
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.scatter(x=df["HC.x"], y=df["HS.x"], color="green")
plt.title("Correlation between home shots and corners", fontsize=14)
plt.xlabel("Number of home corners", fontsize="14")
plt.ylabel("Number of Home shots", fontsize="14")
plt.show()

# create the graph showing the correlation between goals scored and the team xg (figure 1)
f3 = plt.figure(figsize=(19, 15))
ax = sns.boxenplot(data = df, x = df['scored'] , y = df['xG'], hue=df['scored'])
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel(xlabel="Number of goals scored", fontsize=14)
plt.ylabel(ylabel="Team XG", fontsize=14)
plt.title("Goals Scored vs Team xG", fontsize=14)
plt.xlim([-1, 6])
plt.ylim([0.02, 6])


# create the training and testing data
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.4)
train_labels = train.iloc[:,-1]
train_data = train.drop(['result'], axis=1)
test_labels = test.iloc[:,-1]
test_data = test.drop(['result'], axis=1)
train_labels_mc = train['result']
test_labels_mc = test['result']

# Create the Deep Neural Network and build the model with the nodes
model = keras.models.Sequential([
keras.layers.Flatten(input_shape=[29]),
```

```
keras.layers.Dense(128, activation="relu", kernel_initializer='normal'),
keras.layers.Dropout(rate=0.3),
keras.layers.Dense(128, kernel_initializer='normal',activation='relu'),
keras.layers.Dropout(rate=0.5),
keras.layers.Dense(64, kernel_initializer='normal',activation='relu'),
keras.layers.Dense(3, activation='linear')
])
model.summary()

# Compile the model with the correct loss, optimizers, and metrics
import keras.optimizers
model.compile(loss="mean_squared_error",
optimizer="adam",
metrics=["mean_squared_error", "mean_absolute_error"])

# fit the training and testing datasets to the dnn model
history = model.fit(train_data, train_labels_mc, epochs=100, validation_data=(test_data, test_labels_mc), verbose=1)

from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

# create the linear regression model and fit the data
reg = LinearRegression()
reg.fit(train_data, train_labels_mc)
# predict the testing set
matchResultLR = reg.predict(test_data)
from sklearn.metrics import mean_squared_error, mean_absolute_error
# print out the mean squared error and mae
print(mean_squared_error(test_labels_mc, matchResultLR))
print(mean_absolute_error(test_labels_mc, matchResultLR))

from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDRegressor
# create the sgd regressor model
sgdreg = make_pipeline(StandardScaler(), SGDRegressor())
# fit the data
sgdreg.fit(train_data, train_labels_mc)
# predict the data
sgd = sgdreg.predict(test_data)#
# print out the mse and mae
print(mean_squared_error(test_labels_mc, sgd))
print(mean_absolute_error(test_labels_mc, sgd))

# create the graph to show the performance of the regression model (figure 5)
fig = plt.figure(figsize = (9, 9))
ax1 = fig.add_subplot(111)
# Create the actual scatter plot
ax1.scatter(test_labels, test_labels_mc, s=9, c='red', marker="o",
label='Actaul Dataset Data')
ax1.scatter(test_labels, matchResultLR, s=9, c='green', marker="s",
label='Regression Model')
plt.legend(loc='upper right')
plt.xlabel('Match')
plt.ylabel('Win (0), draw (1), loss (2)')
plt.title('Linear Regression Model vs Dataset Data')
plt.legend()
plt.show()

# create the graph to show the bar graph with the 3 models comparison (figure 6)
```

```
X = ['DNN','Linear Regression','SGD Regressor']
Y = [0.0150,1.091e-29,0.0024037]
Z = [0.0852, 2.58e-15,0.0340784]
X_axis = np.arange(len(X))
# make the bar graph
plt.bar(X_axis - 0.2, Y, 0.4, label = 'Mean Squared Error')
plt.bar(X_axis + 0.2, Z, 0.4, label = 'Mean Absolute Error')
plt.xticks(X_axis, X)
plt.xlabel("Model")
plt.ylabel("Score")
plt.title("Model MSE and MAE Performance")
plt.legend()
plt.show()

# make the chart showing the performance of the linear regression model with the training set
fig = plt.figure(figsize = (9, 9))
plt.scatter(train_labels, train_labels_mc, color = "red", label="Training data", s=9)
plt.scatter(train_labels, reg.predict(train_data), color = "green", label="LR prediction", s=9)
plt.title("Predicting Results (Training set)")
plt.xlabel("Match")
plt.ylabel("Result")
plt.legend(loc='upper right')
plt.show()

# make the bar graph showing the performace of the linear regression model (figure 8)
X = test_labels[10:30]
Y = test_labels_mc[10:30]
Z = matchResultLR[10:30]
X_axis = np.arange(len(X))
fig8 = plt.figure(figsize = (30, 15))
# make the bar graph
plt.bar(X_axis - 0.2, Y, 0.4, label = 'Testing data')
plt.bar(X_axis + 0.2, Z, 0.4, label = 'Linear regression')
plt.xticks(X_axis, X, rotation=45)
plt.xlabel("match")
plt.ylabel("Result")
plt.title("testing dataset ")
plt.legend()
plt.show()
```